**Preprocessing:**

* We have start working on preprocessing part of the DRIVE dataset.
* In case of preprocessing of images What we have done is:
* First, we applied masking on the image and stored the images in masking folder.
  + Code for masking can be found here: [MASKING](https://docs.google.com/document/d/1Me54qruQa-zCEOixVGLIRmNWVvN76hcQDEbwBuncaqI/edit?usp=sharing)
* After masking of image, we applied CLAHE on each image for contrast enhancement of vessel structure and stored the CLAHE enabled images in folder.
  + The parameters selected are
    - clipLimit = 2.0
    - tileGridSize = (8,8)
  + Image is first converted to LAB format for CLAHE implementation on each channel and than merged again for final image
  + Code for CLAHE implementation can be found here: [CLAHE](https://docs.google.com/document/d/1xJjaqNuaM44GZdI25I1O0ICc-EF9bkP0ub4aDxwgh6I/edit?usp=sharing)
* After ‘CLAHE’, we applied Non-Local Mean to the image to remove the noise generated during CLAHE and stored these images in a folder.
  + Common arguments are:
    - h: parameter deciding filter strength. Higher the h more removel of noise but more loss of data. 3 is selected
    - h For Color Components: same as h, but for color images only. Again 3 is selected.
    - templateWindowSize: should be odd. 7 is selected
    - searchWindowSize: should be odd. 21 is selected
  + Code can be found here: [Non Local Mean](https://docs.google.com/document/d/1dsXqxKs1sA3324up4kDfaoJoo8hp5d2pQbqHQVSMmGA/edit?usp=sharing)

All parameter selection is done on experimental basis

* There were some problems generated during the entire process of preprocessing.
  + In images like in number ‘34’ and ‘23’ image, after application of Non-Local Mean some vessel structures were erased from image.

**Patch Creation:**

* Loading Images:
* First, we have to load the images, i.e. we have to make a numpy array of shape (20,584, 565,3)
  + 20 refer to number of images
  + 584 by 565 size of each image
  + 3 is number of channels representing the R, G and B channel.
* **Use of Glob.glob:**
* We did this using Glob.glob() function which is taking random images and concatenating the images to from above array, which is major **problem.**
* **Things Discovered**
  + - * For purpose of patch extraction After preprocessing of image, we have three option to discover:
        + **Rule1**: We can create 25\*25 patch with proper overlapping type structure.
        + **Rule2**: We can create 25\*25 patches but with overlapping gap of some pixels to avoid overlapping

Here we reduced the processing to avoid memory exceed error.

But due to the gap of 11 pixels we got only 56720, which is very less in number

So we finally decided to skip 2 pixels on both sides i.e. horizontally and vertically

* + - * + **Rule3**: We can reduce the patch size to 17\*17 or 19\*19 (haven’t tried yet).

**ACC. to Rule 1:**

* Step 1: Separate channels: Input array shape (20, 584, 565, 3)
  + We first have to separate the input array into 3 different channels RED, GREEN, BLUE.
  + RED, GREEN, BLUE SHAPE: (20, 584, 565)
* Step2: Create Patches (Rule 1): input shape (20, 584, 565, 3)
  + We can create patch of 3 different channels separately with each channel shape of flattened array (6599200, 625, 1), there are total 6599200 patches acc. to rule1.
* Step3: Combining Patches:
  + For Each and every patch of every single image All separated channels will now be combined by appending(np.append) each channel and got shape (3,625, 1).
  + As there are 6599200 patches in total, therefore output shape we get is (6599200,3,625,1).
* Step 4: Reshape patches:
  + Input Shape: (6599200,3,625,1)
  + Output Shape: (6599200,3,25,25).
* Problems faced during rule1:
  + While combining patches we initially were using np.append( ) function for appending and combining the three channels,
    - Due to this in resulting shape the channel was at the first like (3, 25, 25).
    - But we want shape to be (25, 25, 3).
* To solve this problem, we used cv2.merge() function for appending and combining the three channels resulting shape was (625, 1, 3).
* **Another Problem is Memory limit exceeds**
* This problem is solved by reducing the patch size 25 to 9

For patch size 25\*25 it takes ~8 hrs /channel to extract patches and memory limit error after ~13 hrs

For patch size 9\*9 it takes nearly ~8 min/channel to extract patches and completed ~20 min

* Step 5: Output Array (i.e. output corresponding to each patch)
  + Firstly all the images are loaded in single list and shape is (20,584,565) as all output images i.e. 1st manual is of single dimension and contain binary values of 0 and 255
  + As there is pure overlapping so by simply flattening the array of (20,584,565) we got a 1-D array of 6599200

**ACC. To Rule 2:**

* Step 1: Separate channels: Input array shape (20, 584, 565, 3)
  + We first have to separate the input array into 3 different channels RED, GREEN, BLUE.
  + RED, GREEN, BLUE SHAPE: (20, 584, 565)
* Step2: Create Patches (Rule 2) with skipping 2 pixels: input shape (20, 584, 565, 3)
  + We can create patch of 3 different channels separately with each channel shape of flattened array (1652720, 625, 1), there are total 1652720 patches acc. to rule 2.
  + No. of patches can be calculated by formula below
* Step3: Combining Patches:
  + For Each and every patch of every single image All separated channels will now be combined by merging (cv2.merge) each channel and got shape (625, 1, 3).
  + As there are 1652720 patches in total, therefore output shape we get is (1652720,625,1,3).
* Step 4: Reshape patches:
  + Input Shape: (1652720,625,1,3)
  + Output Shape: (1652720,25,25,3)

We select to skip 2 pixels by experimenting.

We have skipped 11 pixels in start but we got only 56500 patches from 20 images which is very less data. The codes can be found below

[25n25nskip11pixels](https://docs.google.com/document/d/1MMizwyP9pMhN1rjtezLaTl8LDnZ1BZniud6iorBGvZo/edit?usp=sharing) (Time taken in nearly 10 mins in total)

[25n252skip2pixels](https://docs.google.com/document/d/1hbxMVZjDcQ1iMm4g9-Wbg8RC8xvvBxFOMcK2gV3VEG8/edit?usp=sharing) (Time taken is 50 min/channel & in total 2:30 hrs)

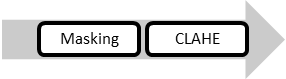
* Step 5: Output Patch Array (i.e. output corresponding to each patch)
  + Firstly, all the images are loaded in single list and shape is (20,584,565) as all output images i.e. 1st manual is of single dimension and contain binary values of 0 and 255
  + As there is skipping of 2 pixels so by simply skipping 2 pixels in the array of (20,584,565) we got a 1-D array of 1652720
  + Code can be found here: [Output](https://docs.google.com/document/d/1oIeq9aWMY8rE82POuVJmrJ0DG5PzFwlTNJnMy3g6-n8/edit?usp=sharing)
* Step 6: We Save the patch array for further use.

DRIVE DATASET

Pre-Processing

Method 1

Method 2



Patch-Extraction

×

[RULE 1](#Rule1)

Patch-Extraction

[RULE 2](#Rule2)

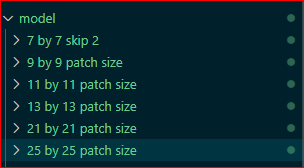
×

×

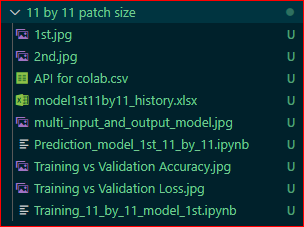
* Step 7: Get the Output array (1st manual images): By skipping the step by 2 create the output array of “1st manual” images and save this output array also.
  + Size of output array: (1652720, 1)

**Training (Using Rule 2)**:

* **Directory Structure of Model 1:**



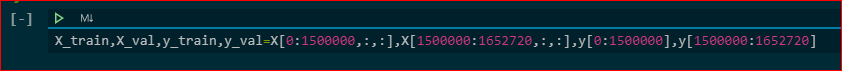
* + Using Rule 2 We trained our model by experimenting on patches of different sizes, to check how the training and validation accuracy vary on different patch sizes.
  + The more is the patch size it takes more computation time to extract the patch array.
  + Therefore, our goal is to:
    - patch size to be used for training should be as much less as much possible, but without compromising with the training accuracy.
  + Total number of patches Formed: 1652720
  + We experiment on following patches:
    - 7 by 7
    - 9 by 9
    - 11 by 11
    - 13 by 13
    - 21 by 21
    - 25 by 25
  + **Directory Structure of model by taking 11 by 11 patch size:**



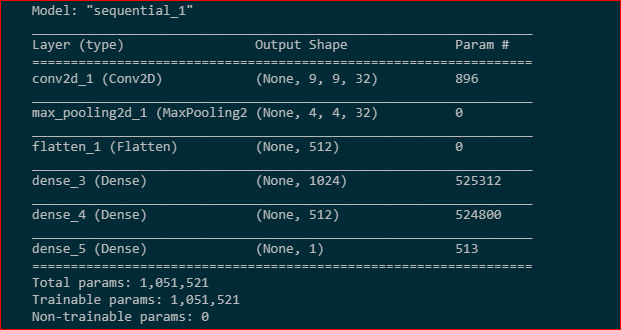
* + “Training\_11\_by\_11\_model\_1st.ipynb “is the python file in which code for training of model is written.
* **Training Process:**
* **1st training**:
  + First load both the output numpy arrays which we saved ([step 6](#Rule2), [step 7](#Rule2)) of patches created.

X = output array 1(1652720, 25, 25, 3), y = Output array 2(1652720, 1)

* + - Shape of output numpy array1: (1652720, 25, 25, 3)
    - Shape of output numpy array1: (1652720, 1)
  + Splitting the Output array:
    - During 1st training of our model 1st we split our training data into two-two subarrays of both X and y, so that we can check for training accuracy and validation accuracy.



* + Develop a keras Convolution Model:



* + Train the model using the above, mentioned model summary.
  + **Training accuracy acc. to different patch sizes (10 epochs):**

|  |  |  |
| --- | --- | --- |
| Patch size | Training Accuracy | Training loss |
| 7 by 7 | 0.9560 | 0.1238 |
| 9 by 9 | 0.9593 | 0.1125 |
| 11 by 11 | 0.9607 | 0.1097 |
| 13 by 13 | 0.9603 | 0.1136 |
| 21 by 21 | 0.9559 | 0.1302 |
| 25 by 25 | 0.9135 | 0.2943 |

* + As we can observe from above table, we concluded that we will further choose 11 by 11 size patches it is taking less computation and also giving us better training results.